DRAFT

Software Design Document for the Land Information System

Submitted under Task Agreement GSFC-CT-2

Cooperative Agreement Notice (CAN) CAN-00OES-01

Increasing Interoperability and Performance of Grand Challenge Applications in the Earth, Space, Life, and Microgravity Sciences

Version 1.0

DRAFT

Revision history:

Version	***Summary of Changes***	***Date***
1.0	Initial release.	7/11/02

Table of Contents

List of Figures

List of Tables

1. Introduction

1.1 Identification

This Software Design Document establishes the software design for the Land Information System (LIS). LIS is a project to build a high-resolution, high-performance land surface modeling and data assimilation system to support a wide range of land surface research activities and applications.

This document has been prepared in accordance with the requirements of the Task Agreement GSFC-CT-2 under Cooperative Agreement Notice CAN-00-OES-01 Increasing Interoperability and Performance of Grand Challenge Applications in the Earth, Space, Life, and Microgravity Sciences, funded by NASA's ESTO Computational Technologies (formerly High Performance Computing and Communications) Project.

1.2 Purpose

This document serves as the blueprint for the software development and implementation of the Land Information System (LIS).

1.3 Scope

This document covers the design of all the LIS software components for the three-year duration of the LIS project. The document focuses primarily on theimplementation of the LIS software on a general-purpose Linux cluster system, and most of the component designs also apply to an SGI Origin 3000 system. This document does not cover design for other hardware/software platforms.

2. System Design Overview

2.1 System function

LIS is a complex software system that is designed to perform the following functions:

- Realistic land surface modeling. LIS will simulate the global land surface variables using various land surface models, driven by atmospheric "forcing data" (e.g., precipitation, radiation, wind speed, temperature, humidity) from various sources.
- High performance computing. LIS will perform high-performance, parallel computing for near real-time, high-resolution land surface modeling research and operations.

- Efficient data management. The high-resolution land surface simulation will produce a huge data throughput, and LIS will retrieve, store, interpolate, re-project, sub-set, and backup the input and output data efficiently.
- Usability. LIS will provide intuitive web-based interfaces to users with varying levels of access to LIS data and system resources, and enforce user security policies.
- Interoperable and portable computing. LIS will incorporate the ALMA (Assistance for Land surface Modeling Activities) and ESMF (Earth System Modeling Framework) standards to facilitate inter-operation with other Earth system models. LIS will also perform its functions on both generic Linux clusters and SGI Origins with minimal modification of the LIS software.
- System management. To ensure reliable operation of the LIS system, LIS will perform various system monitoring and management duties.

2.2 System architecture

The function of LIS dictates a highly modular system design and requires all the modules, or components, to work together smoothly and reliably. Figure 1 shows an overview of the LIS software architecture and its components, and their interactions. LIS will continuously take in relevant atmospheric observational data to force three different land surface models, and the land surface simulation is carried out in a highly parallel fashion. Meanwhile the large amount of output data will be efficiently managed to facilitate reliable and easy access. Moreover, some of the components and their interfaces, whenever appropriate, will conform to either the ALMA or ESMF standards, or both. Finally, there will be software components to monitor the software and hardware status and manage them to ensure sustained high performance output and high availability.

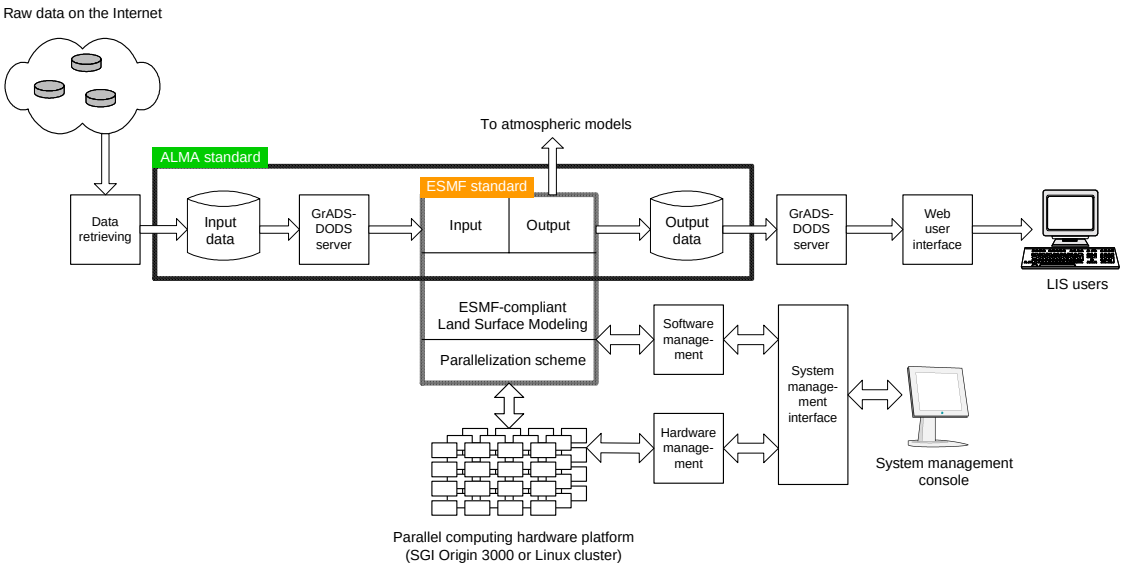

Figure 1: LIS overview and its components.

2.3 System components

From the perspective of software design, the LIS system components are classified into four subsystems: land surface modeling, data management, user interface and system management.

2.3.1 Land surface modeling

Land surface modeling is the core of the LIS system. The land surface modeling components work together to regularly take in the forcing data and feed them into the three land surface models. The output data from the simulation results are saved at regular intervals. Due to intensive computational requirements resulting from high-resolution simulation, the land surface modeling will be implemented as a distributed and parallel architecture, as shown in Figure 1. The performance of the parallel computing will be monitored, analyzed and tuned with the system management software components.

Also shown in Figure 1 are two sets of interfaces for the land surface modeling component of LIS. The ALMA interface standard (http://www.lmd.jussieu.fr/ALMA/) defines “Required”, “Recommended” and “Optional” input and output variables, (including names, units, and sign conventions) and is therefore a high-level interface. The developing ESMF interface standard is expected to provide standard definitions of time and space coordinates, in addition to an I/O API, and as such will serve as a low-level interface for LIS land surface models. Therefore, the land surface modeling interfaces for LIS should be able to be compliant with the standards of both ALMA and ESMF

2.3.2 Data management

Three kinds of data are involved in the LIS system: input data, which are mainly the atmospheric forcing data; output data, the simulated land surface variables; and parameter data, which include static data sets such as vegetation classification data, land masks and soil types.

Data management will be implemented by various components that are responsible for retrieving the input forcing data from various locations over the Internet, and for staging the data in a timely manner to ensure near real-time operation of LIS. Meanwhile, data management components will take care of the output data LIS produces, and satisfy users’ access requests via various access channels. Data management modules also ensure the continuous availability of the parameter data, and efficient archival and retrieval of selected input and output data. Again, as shown in Figure 1, ALMA and ESMF standards will be followed by the data management modules in data file pre-processing, re-projection, interpolation and storage processes.

The data management functionalities will be implemented in a modular fashion as standalone components with well-documented APIs, standalone programs or scripts, and according to the client-server architecture of the GrADS-DODS system.

2.3.3 User interface

LIS will implement a user interface for users' easy and unified access to LIS data and processing power. The user interface will be a largely web-based, three-tier client-server system. GrADS-DODS server and various CGI scripts will be used as the middle tier to accept and process users' requests, and interface with LIS data files or job scheduling processes. Password authentication will be used, and three levels of authorization will be enforced, depending on a user's credentials obtained from the authentication process.

2.3.4 System management

The system management components interact closely with the other software components to ensure high reliability and high availability of the whole system. The system management components will be implemented in a highly structured manner with functions to handle system information from the hardware layer, inter-connect layer, operating system layer, all the way up to the application layer. These modules will use various custom scripts as well as existing open-source software packages and kernel modules, such as SNMP, MRTG, Big Brother and lm_sensors.

3. Land Surface Modeling Design Details

3.1 Function

The land surface modeling components are designed to perform high-performance, parallel simulation of global, regional or local land surface processes with initially three land surface models: the CLM model, the NOAH model and the VIC model. Specifically, the land surface modeling components will interact with the data management subsystem to obtain properly formatted input forcing data, and pass the forcing data, alone with other static parameters, to the three land surface models through the land surface driver component. Each of the land surface model carries out the simulation on a distributed, parallel hardware platform, either a Linux cluster or a SGI Origin 3000. The results are passed to the output component, which interacts with the data management subsystem to handle the output data. The parallelization process is managed by the system management subsystem. The components interface each other in accordance with ALMA and ESMF standards, wherever applicable.

3.2 Architecture

Figure 2 shows the software architecture of the land surface modeling subsystem. The components are designed to be modular with well-defined interfaces that comply with

ALMA or ESMF standards. The interface between the land model driver and three land models, CLM, NOAH and VIC, will comply with ESMF and will be general enough so that additional land surface models can be added without much modification of the code. The land surface modeling subsystem is designed in a way that multiple copies can run as different processes in parallel, independent of each other, with each of them processing a different piece of land surface.

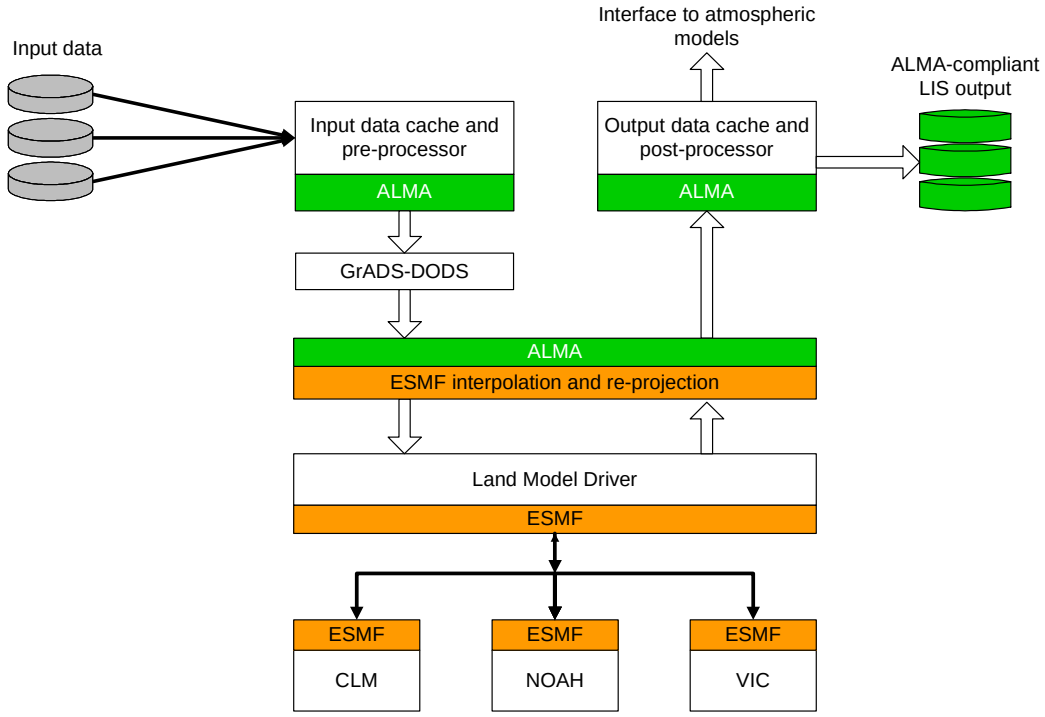

Figure 2: LIS land surface modeling architecture with ALMA and ESMF interfaces.

3.3 Implementation

3.3.1 Portability and interoperability implementation

The land surface modeling subsystem is designed to be running in parallel, both on a Linux cluster with 200 nodes, and on a SGI Origin 3000 platform with 512 processors. Although the hardware architecture differs greatly between the distributed-memory Linux cluster and the shared-memory SGI Origin 3000, our implementation of the land surface modeling programs will make this architectural difference fairly transparent: On the Linux cluster, each node will run a copy of the land surface modeling process; on the SGI Origin, each CPU will run a copy. Thus we establish a direct correspondence between a node in the Linux cluster and a CPU in the Origin 3000, and the hardware architectural differences will not matter to our design of the software; The land modeling scheme will be able to run on both platforms with minimal modification. So in this document whenever we refer to a node in the Linux cluster, it applies equally to a CPU in the Origin 3000.

Interoperability is achieved by following both the ALMA and ESMF standards closely. By following the ALMA standard, the LIS land surface modeling system is guaranteed to exchange data with other land surface modeling systems that are also ALMA-compliant. ESMF standard will allow us to interact with other Earth system models, such as atmospheric models or environmental models with standard interfaces.

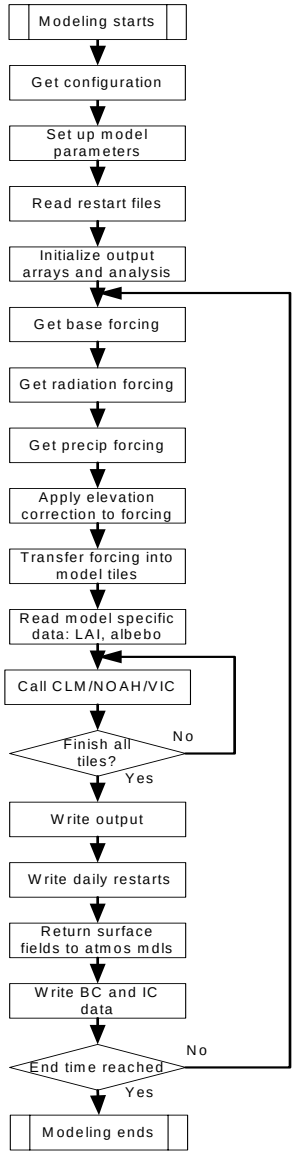

Figure 3: LIS land surface modeling flowchart.

3.3.2 Land surface modeling flowchart

As shown in Figure 3, and described in detail in the land surface model documentation, land surface models proceed in a manner similar to other physical models. Modeling

proceeds given prior knowledge of the spatial and temporal domains of the simulation, in addition to initial conditions and parameters required to solve the equations of water and energy conservation within that domain. Modeling proceeds according to increments of time ("time steps", typically 15 minutes), until the ending time is reached and data is written out for future runs and analysis.

3.3.3 Compute nodes job processing

A compute node's job is straightforward: it runs a copy of the land surface modeling subsystem in its process space, computes a piece of land surface obtained from the master node, and requests another piece of land surface from the master node as soon as it finishes the current piece, until the master node refuses to give it any pieces, in which case there are no more land pieces are available and the compute node's job is done. Figure 4 shows the flow chart of the compute node's job handling process.

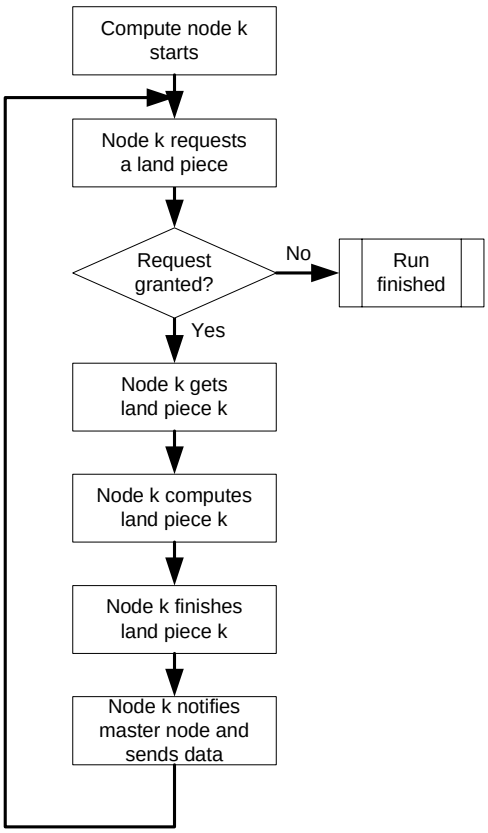

Figure 4: Compute nodes flowchart for parallel computing of land surface modeling.

3.3.4 Parallelization scheme and master nodes job processing

Job volume: We estimate that at 1km resolution LIS will deal with ~50,000 times more grid points than the 2ºx2.5º resolution GLDAS . To satisfy the requirements of real-time

operation, the job, which includes a gridded representation of the global land surface, must be split into smaller pieces and run in parallel. We plan to divide the global surface into 10,000 small land pieces, and with 1km resolution, each piece would require about 5 times as many computations as the 2ºx2.5ºGLDAS , and will take a single computing node about 200MB memory to run, and 10 minutes to finish a 1-day simulation, based on the initial performance baselining of GLDAS running at both 2ºx2.5º and 0.25ºx0.25º resolutions. The Linux cluster can consume approximately 200 pieces per round, and under ideal conditions, it will take the whole cluster about 50 rounds to finish the whole job. This will take 500 minutes, or about 9 hours, to finish a 1-day simulation of the whole global land surface, which satisfies the real-time requirement with enough extra room. We expect that the timings on the SGI Origin will be comparable to those on the cluster, although memory and disk limitations, some imposed by the queue structure, will likely prohibit effective use of that system for demonstrating LIS in a near-real-time mode. However, we plan to demonstrate the LIS on the SGI Origin system as proof-of-concept.

Parallelization paradigm: The design uses the slightly modified version of the “pool of tasks” scheme for the parallel processing of the land pieces. One of the master nodes will keep three tables on hand when starting the job: table of unfinished-jobs, finished-jobs, and jobs-fetched. At the beginning, the 10,000 land pieces are listed in the "unfinished" table, and each compute node comes to the master to fetch a piece from it, and starts working on it. The master node then moves the fetched jobs to the "jobs-fetched" table, and starts a timer for each fetched job. When a compute node finishes a job and notifies the master node before the job’s corresponding timer runs out, this piece is regarded a finished job, and the master node moves it from the "fetched" table to the "finished" table. And the compute node goes on to fetch another job until the "unfinished" table is empty. If a fetched job's timer runs out before the compute node reports back, the master node then assumes that that particular compute node must have crashed, and then moves that timed-out job from the "fetched" table back to the "unfinished" table for other compute nodes to fetch. Figure 5 shows the flowchart (left) of the master node’s job handling and scheduling process, and the various status of the three tables (right) the master node uses to keep track of the job progress at different corresponding stages in the flowchart.

4. Data Management Design Details

4.1 Function

The data management subsystem is composed of the following functions: input data retrieval from the Internet, data pre-processing and post-processing, data interpolation and sub-setting, output data aggregation, storage, backup and retrieval. It links the other subsystem together, and ensures smooth end-to-end data flow, from the input raw data all the way to the output data satisfying LIS users’ various requests.

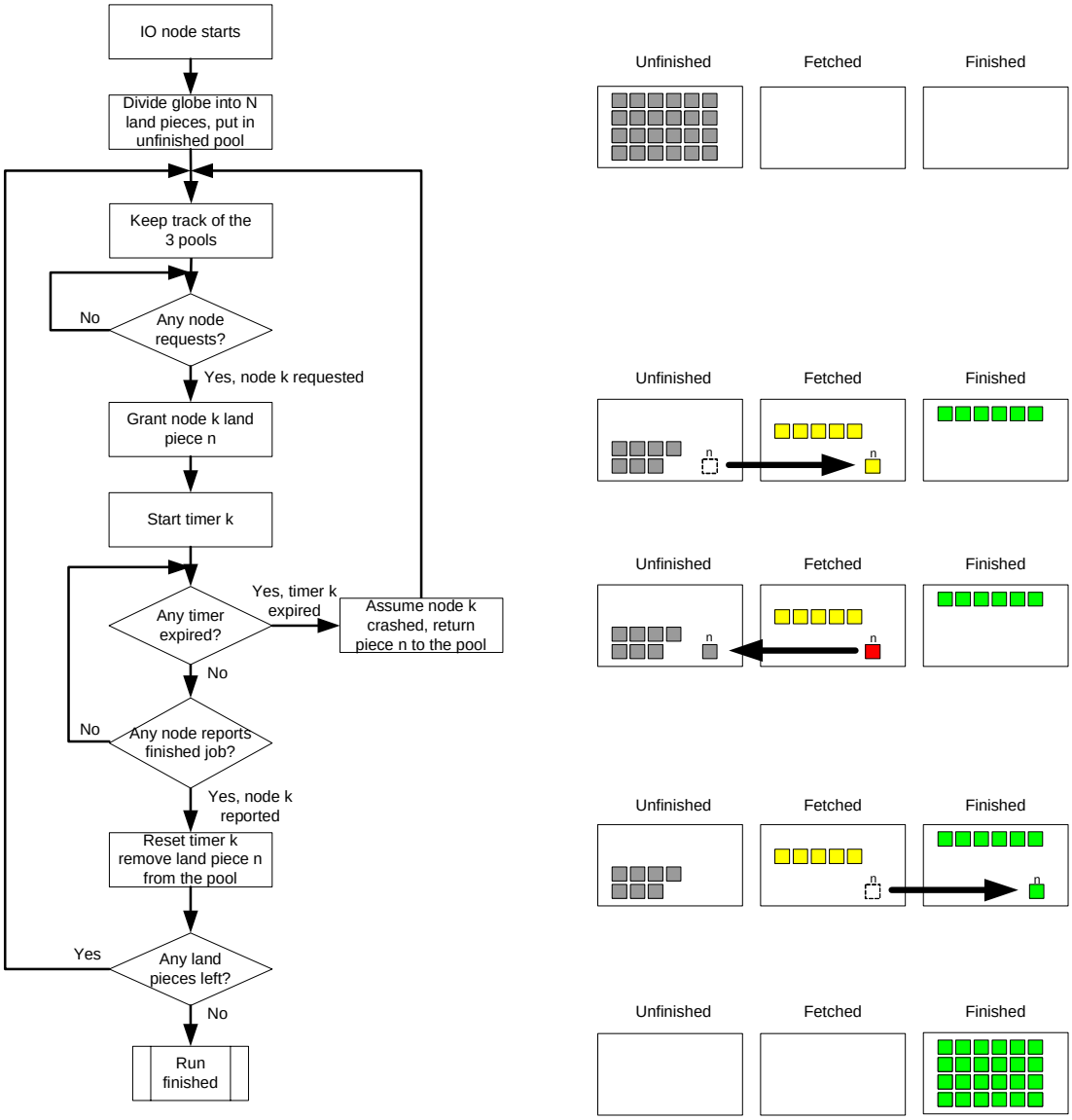

Figure 5: Parallel computing control flowchart (left) and parallelization scheme (right) of an IO node.

4.2 Data flow design

Figure 6 shows the logical data flow of LIS system. LIS data management will deal with three categories of data: static data, input forcing data and output data. In the future when data assimilation is incorporated, there will be observational data added to the input data.

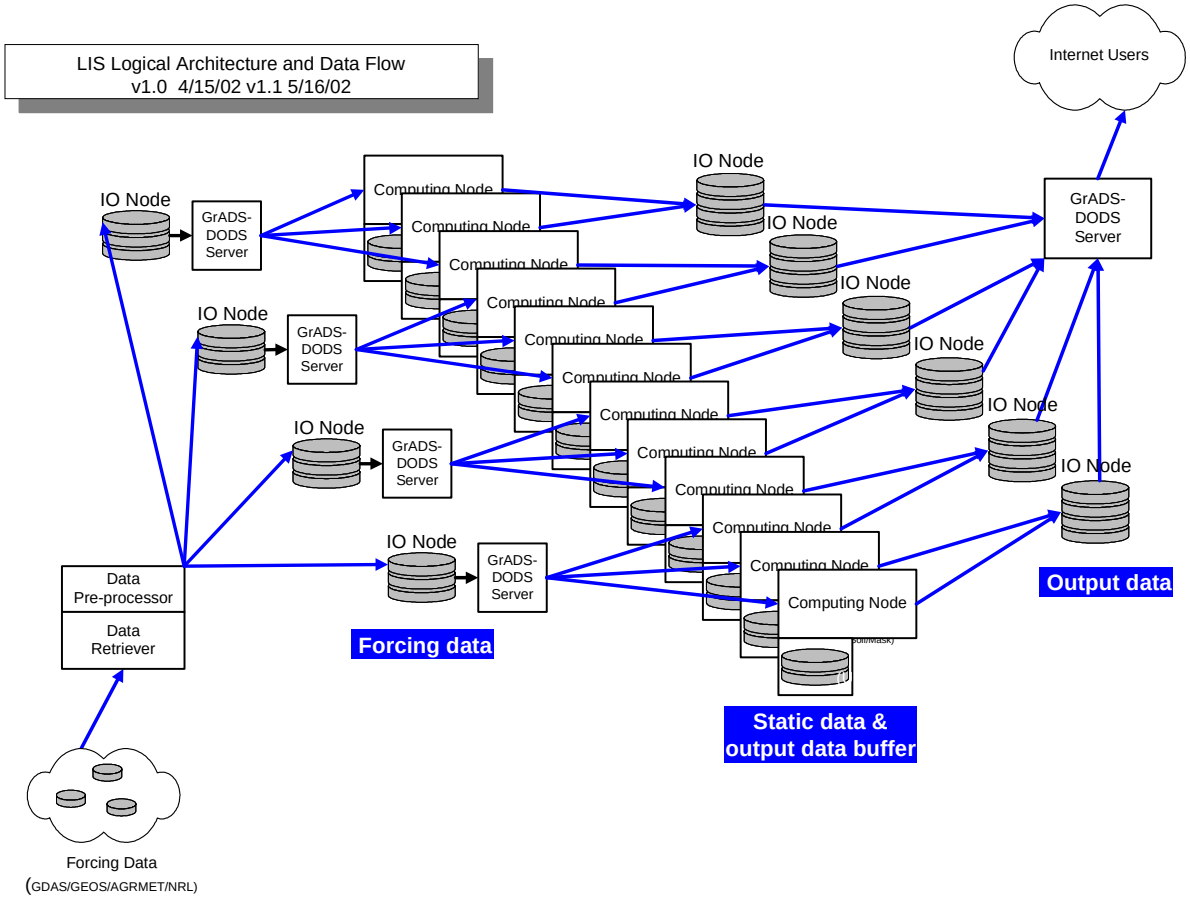

Figure 6: LIS logical architecture and data flow.

The static data include the vegetation classification, land mask, etc., with an estimated size of 136 GB. Since these data will not be updated frequently, we want to put a copy of these data on each compute node's local disk to reduce network traffic. With some optimization, we expect all of the static data will fit on the node's 80 GB disk.

The forcing data, fetched from various locations on the Internet, need to be fed to the compute nodes at regular intervals. The total traffic is estimated to be 279 MB/day, which is not significant compared to the output data traffic. We designate one of the master nodes to fetch and pre-process the data, then send a copy of the forcing data to the other master nodes via NFS system. When a compute node needs the forcing data, it will contact the master node, which corresponds to the sub-cluster it belongs to without bothering other master nodes. To further reduce the IO network traffic, each master node will run the GrADS-DODS server to feed the compute nodes with the sub-set of the data they need.

The output data will be stored on the master nodes too, and served to users via a GrADS-DODS server running on one of the master nodes. Since it is not feasible to store the output in a single file (200 GB/day), we want to distribute the data across all the master nodes. To keep the huge output data volume manageable, we designed a storage scheme that will distribute the land surface variables in the output data across the master

nodes. Since there are 40-48 variables in the output data, with some of them having multiple levels, we can let each master node to store the global data of only 6 or so of the output variables. So on average, the I/O traffic is segregated and each master node is only taking 1/8 of the total data traffic, and the subsequent operations by the GrADS-DODS servers are greatly simplified.

4.3. Data files

Table 1 lists all the data files and specifications involved in the data management subsystem.

Table 1: LIS data files

LIS Data Files and Estimated Data Volume for LIS with 1/100 X 1/100 (~1km X 1km) Resolution, based on the data used for GLDAS 1/4 X 1/4 (4/12/02).					
Dataset	***Tentative name/location***	***Desired/native resolution***	***Native format***	***Approx size***	***Update frequency***
UMD Vegetation classification map	GVEG/UMD_60G0.01.txt	1/100 X 1/100	ASCII	65G	Static
UMD Land mask	GVEG/UMD60mask0.01.asc	1/100 X 1/100	ASCII	18G	Static
Soil classification map	BCS/sim60soil0.01.txt	1/100 X 1/100	ASCII	20G	Static
Soil color map	BCS/soicol60.01.bin	1/100 X 1/100	Binary	2G	Static
Sand fraction file	BCS/sand60.01.bfsa	1/100 X 1/100	Binary	6G	Static
Silt fraction map	BCS/silt60.01.bfsa	1/100 X 1/100	Binary	6G	Static
Clay fraction map	BCS/clay60.01.bfsa	1/100 X 1/100	Binary	6G	Static
Porosity map	BCS/por60.01.bfsa	1/100 X 1/100	Binary	6G	Static
Slope map	BCS/slope60.01.bfsa	1/100 X 1/100	Binary	2G	Static
Koster tile space file	GVEG/tile_info.0.01	TBD			
Leaf area index (LAI)	LDAS/BCS/lai.dat			1M	
AVHRR-derived LAI climatology	/GLDAS5/DATA/AVHRR_LAI			5G	
Static file size				**136G**	
NCEP GDAS Forcing data file	/GLDAS4/DATA/GDAS/	Native T170, ~0.7deg	GRIB	50M/day (3.2M X 4 X4)	Every 6 hours
GEOS forcing data	/GLDAS4/DATA/GEOS/BEST_LK	1 deg	Binary	25M/day	Every 3 hours
AGRMET SW flux data	/GLDAS5/DATA/AGRMET/SWDN	~48km		48M/day	Every 1 hour
AGRMET LW flux data	/GLDAS5/DATA/AGRMET/CloudAGR			144M/day	Every 1 hour
NRL Precipitation data	/GLDAS1/DATA/NRL	1/4 degree		12M/day	Every 6 hours
Total data input flux				**279M/day**	
Estimated Output Data Volume for LIS (1/100 X 1/100) Resolution, Based on GLDAS Runs with 2 X 2.5 Resolution. Data output interval is assumed to be the same.					
CLM output data	OUTPUT	1/100 X 1/100	GRIB	200G/day	
				(5 times more if in binary format)	
Total data output flux				**200G/day**	

4.4. GrADS-DODS server

GrADS-DODS servers will be employed both to serve the input data to the land surface computing code, and to serve the output to the Internet users. A GrADS-DODS server uses a typical client-server architecture to communicate with the DODS clients. The communication protocol between a client and a server is HTTP. A GrADS-DODS server has two parts, the front end Java servlets contained in Tomcat and the back end GrADS running in batch mode. The HTTP queries are processed by the front end Java servlets, managed by Tomcat application server. Data-retrieving, sub-setting and processing on the server side are performed by the GrADS engine.

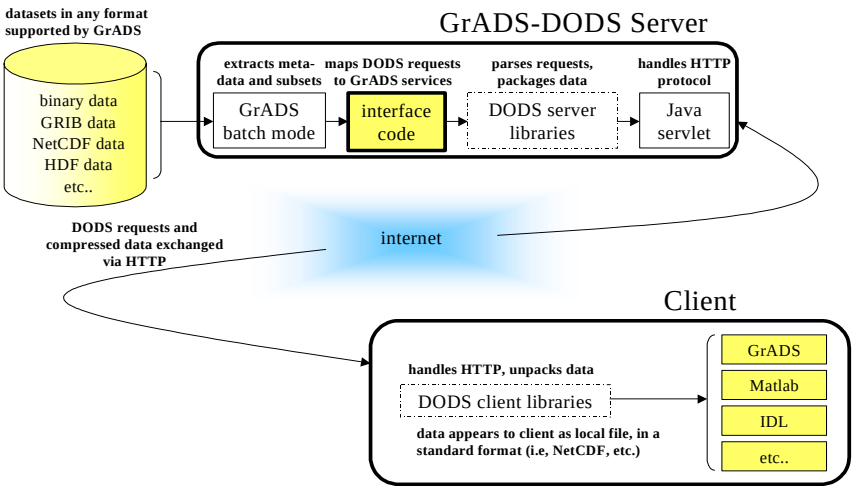

Figure 7: GrADS-DODS client-server architecture.

5. User Interface Design Details

5.1 Function

The user interface subsystem provides users with friendly and intuitive channels to access LIS data products, to perform analysis of the data, to employ the LIS processing power for their own model studies. The user interface also enforces user access security policies, and isolates the backend system from the Internet for better security.

5.2. Architecture

The user interface subsystem takes a typical multi-tier client-server system architecture. On the client side, a user has three types of client programs to use as the front-end: a web browser, a ftp client program (which can be integrated in a web browser), or a DODS client program. On the server side, a general purpose web server will be used to serve clients with a web browser, and a GrADS-DODS server will be deployed to serve DODS clients, and a FTP server to server ftp clients. Besides theses components, CGI scripts and CGI-GrADS gateway scripts will be used as the middleware to perform dynamic processing based on users' interactive requests sent through web browsers. Figure 8 shows the user interface architecture design.

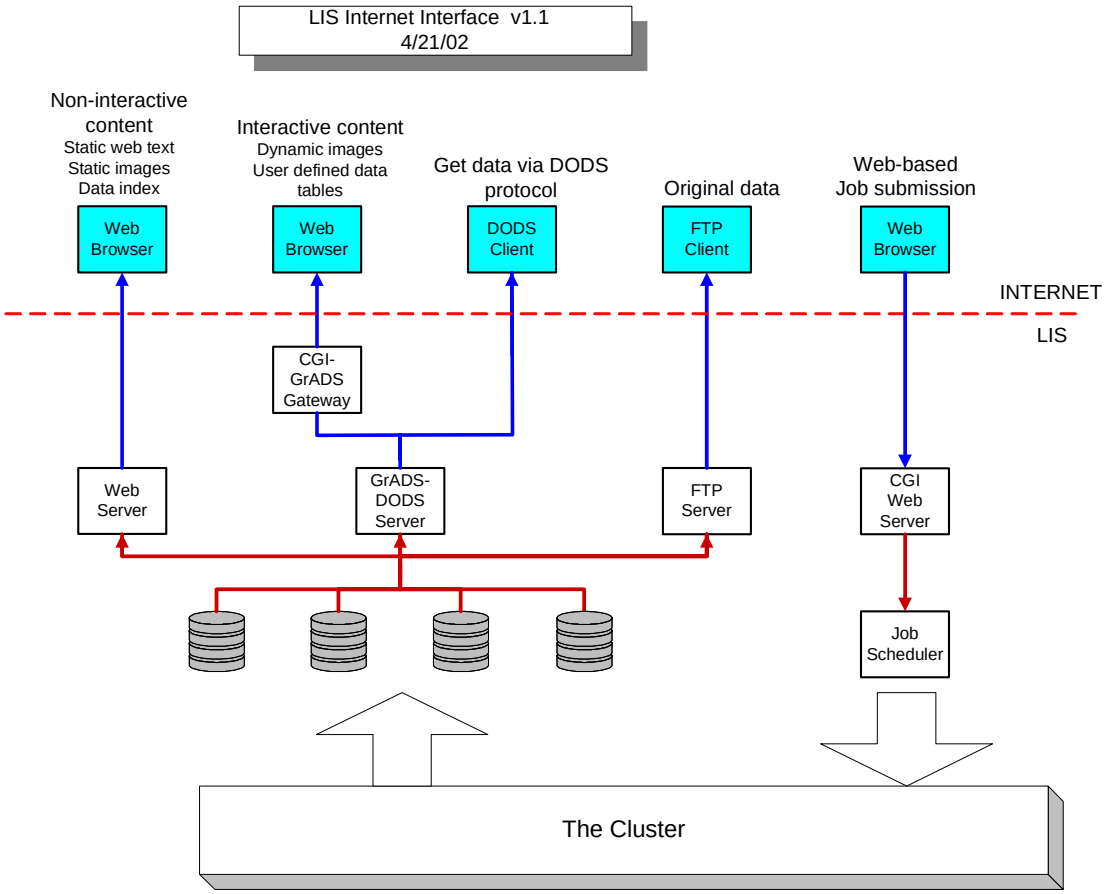

Figure 8: LIS user interface architecture.

5.3 User levels and security design

Outside users accessing the LIS are categorized into three levels, associated with different levels of data access and security requirements.

Level 1 users are the general public, who will access the LIS data primarily through a standard web browser. Information provided to this class includes static images and text, and some limited interactive content such as GIF/JPG/PNG images generated on the fly in response to users' regulated web input. The static content, most of which is static html

pages, is served via the web server, while the interactive content is generated via a three-tier architecture with server-side GrADS as the image engine and below it the GrADS-DODS server as the data engine to feed the server-side GrADS. This group of users does not have direct access to the data or LIS scientific computing power system, and their usage of system resources is very limited. Therefore, for this class of users we do not enforce any additional authentication or authorization procedures. It is also our intention to facilitate easy access to the data for education and outreach purposes.

Level 2 users have direct access to LIS data, either through our GrADS-DODS server by using a DODS client, or directly through ftp fetches. The GrADS-DODS server provides the users with the ability and flexibility to get only a sub-set of the data they need. To be authorized as Level 2 users, they will have to register with us first by filling out web forms, and they will be authenticated using password and source IP addresses before accessing the data. The GrADS-DODS server will impose a limit on system resource usages.

Level 3 users have the highest access level: in addition to all the Level 2 and Level 1 access privileges, they will be able to access the parallel computing power of LIS system. They will be able to submit their jobs, and test their land models, etc., through a web interface. A number of CGI scripts will be interfacing the web input and the LIS system's job scheduler. The number of these users will be limited, and the authorization and authentication process will be enforced in compliance with NASA's and GSFC's relevant regulations.

6. System Management Design Details

6.1. Function

The system management subsystem is responsible for monitoring, maintaining and administering the LIS system to ensure its reliable operation and optimal performance output.

We categorize the system management function into four levels: hardware level, interconnect level, operating system level and application software level. For the SGI Origin 3000 platform, we are not involved in the management of the hardware and interconnect levels. But for the Linux cluster, the hardware and interconnect level management is our responsibility and is critical to the overall stability and performance of the LIS system.

The hardware level system management involves power-up and shutdown of the nodes, booting strategy and hardware status monitoring. Interconnect level management requires the monitoring of the link status of the network nodes, bandwidth usage and traffic statistics. Operating system level management takes care of system resource usages, such as CPU, memory and disk space usage. Application level management oversees the progress of the LIS jobs, configures different runs, analyze performance bottlenecks, and obtain performance profiles for fine-tuning.

6.2 System monitoring data

The following table summarizes the system data of various levels the management subsystem is designed to collect and analyze.

Table 2: System monitoring and management data collection

LIS Cluster System Monitoring and Management Data		
Category	***Data Items***	***Update frequency***
Application level	`Overall cpu/mem of each process`	1min
	`Overall progress of whole job`	2min
	`Progress of each process`	1min
	`Timing of each module`	sampled, off-line
	`Memory usage of each module`	sampled, off-line
Operating system level	`Total memory usage & biggest user`	2min
	`Total CPU usage & biggest user`	2min
	`Total disk space usage`	2min
	`System up-time and running procs`	2min
Interconnect level	`Bandwidth usage of each node`	2min
	`Bandwidth usage of switches`	2min
	`Latency measurements`	2min
	`Packet drops measurements`	2min
Hardware level	`Fan speeds`	10min
	`Chasis temperature`	10min
	`Power supplies voltage`	10min

6.3 Architecture and implementation

The variety of system variables and management duties requires us to design a management system with modules performing individual and well-defined tasks. Figure 9 shows the structured design of the system management functionalities, with the Linux cluster platform as the example.

On the hardware level, we will design scripts to take advantage of the “Wake-on-Lan” technology for powering up the nodes smoothly in a well-defined pattern. The nodes will be able to boot across the network with the PXE technology, as well as from the local disk, to centralize system software management. After booting, each node’s hardware parameters, such as CPU temperature, cooling fan speeds and power supply voltages, will be collected by kernel modules called “lm-sensors”, and sent to the central management station with web-based display with automatic updates.

On the interconnect level, we will use SNMP protocol as the underlying data collection and management mechanism, interfaced with MRTG for web-based display of network statistics. Additional network data can also be collected by Big Brother system and network monitor, also with web output.

On the operating system level, we will use SNMP and various OS shell commands and utilities to collect system data, and use MRTG and Big Brother as the interface.

On the application level, we will develop CGI scripts, interfaced with OS commands and utilities, to provide a web-interface for the monitoring and control of LIS jobs and processes. Standard performance profiling and debugging tools will be used off-line to analyze sample runs for trouble shooting and performance fine-tuning.

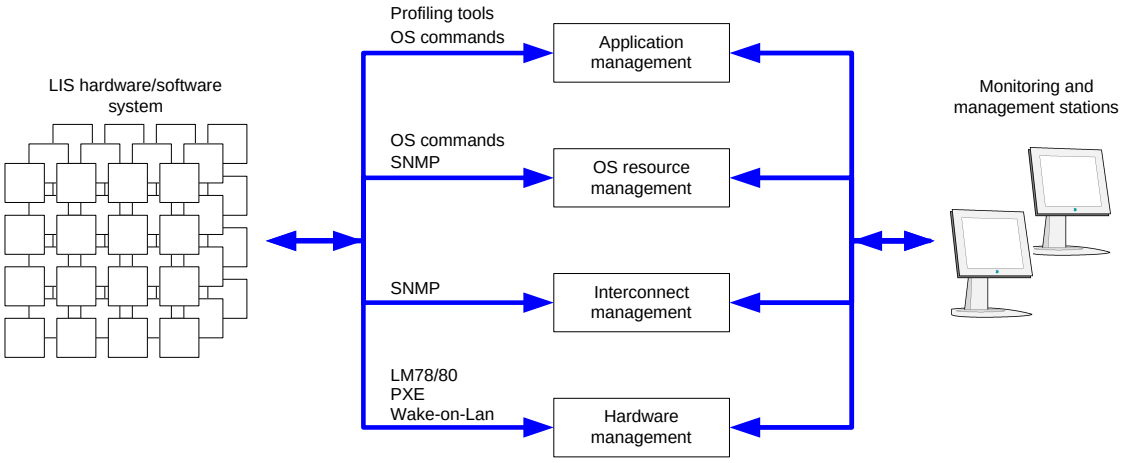

Figure 9: LIS system monitoring and management architecture.

7. References

ALMA: http://www.lmd.jussieu.fr/ALMA/

CLM: http://www.cgd.ucar.edu/tss/clm/

ESMF: http://www.esmf.ucar.edu/

GrADS-DODS server: http://grads.iges.org/grads/gds/

LDAS and GLDAS: http://ldas.gsfc.nasa.gov/

NOAH: http://www.emc.ncep.noaa.gov/mmb/gcp/noahlsm/README_2.2.htm

“Pool of tasks”: H.P. Hofstee, J.J. Likkien, and J.L.A. Van De Snepscheut: "A Distributed Implementation of a Task Pool". Research Directions in High-Level Parallel Programming Languages, pp 338--348, 1991.

VIC: http://www.hydro.washington.edu/Lettenmaier/Models/VIC/VIChome.html

8. Acronyms and Terms

ALMA: Assistance for Land-surface Modeling Activities

API: Application Programming Interface

CGI: Common Gateway Interface

CLM: Community Land Model

DODS: Distributed Ocean Data System

ESMF: Earth System Modeling Framework

GLDAS: Global Land Data Assimilation System

GrADS: Grid Analysis and Display System

LDAS: Land Data Assimilation System

LIS: Land Information System

MRTG: Multi Router Traffic Grapher

NFS: Network File System

NOAH: National Centers for Environmental Prediction, Oregon State University, United States Air Force, and Office of Hydrology Land Surface Model

PXE: Preboot Execution Environment

RAID: Redundant Array of Inexpensive Disks

SNMP: Simple Network Management Protocol

VIC: Variable Infiltration Capacity Land Surface Model